

Bedienungsanleitung für die Matlab Programme

Maximilian Urmann,

20.12.22

1 Testskripte

Als ausführbare Programme dienen die `time_stepping_[...].test` Skripte. Es gibt je eins für jeden Typ von Randwertbedingung, also jeweils ein Skript für homogene Neumann- (HN), Allen-Cahn- (AC), Liu und Wu- (LW) und Goldstein, Miranville und Schimperna- (GMS) Randwerte. Im ersten Abschnitt wird das Gebiet Ω durch `distmesh` übergeben. Dabei wird zeitgleich das Gebiet triangulisiert. Des weiteren erhält man mit `N_Omega` die Anzahl der Freiheitsgrade.

Als nächstes extrahiert das Programm die Randknoten aus den inneren Knoten und bildet daraus die Randelemente. Dabei gibt es für verschiedene Gebiete verschiedene passende Codepassagen.

Im nächsten Abschnitt werden die Startwerte `alpha_0` gebildet. Dabei stehen erneut mehrere zur Auswahl.

Jetzt berechnet `assembly_bulk` und `assembly_bulk` die Massen- und Steifigkeitsmatrizen für das Innere und den Rand.

Abschließend werden noch alle Koeffizienten festgelegt, bevor man alles nötige an die Lösungsfunktion `time_stepping_[...]` übergibt. Wie bereits bei den Skripten existiert hier für jedes Randwertsystem eine eigene Lösungsfunktion. Diese liefert die Lösung die im Anschluss an gewünschten Zeitpunkten geplottet werden kann oder in einem Video dargestellt werden kann.

2 Berechnungsfunktionen

Die Berechnungsfunktionen setzen das time-stepping Schema um. Dazu berechnen sie im Voraus alle wichtigen Matrizen, unter anderem auch die aus den Spuroperator resultierenden Matrix `B`. Diese erhält man durch die Funktion `assembly_B`. Es wird Masslumping verwendet. Die entsprechenden Matrizen generiert `assembly_ML` bzw. `assembly_ML_HN1` für das System mit homogenen Neumannrandwerten.

Nun wird die Sparse-Matrix `K` zusammengebaut, welche im Anschluss wichtig für das nichtlineare Gleichungssystem ist. Dies geschieht Gleichung für Gleichung.

Nun initialisiert man noch die Lösungsmatrizen, welche im Folgenden durch die berechneten Lösungen ergänzt werden können bevor man mit der Berechnung im Loop startet.

In der Schleife holt man sich zu Beginn die Lösung zum vorherigen Zeitschritt und speichert diese in `alpha_n` und `xin`. Daraus bildet man nun im Folgenden die Teile der Funktion die durch den vorherigen Zeitschritt bereits gegeben sind. Daraus resultiert dann letztendlich der Vektor `b`.

Jetzt definiert man die Funktion, von der man die Nullstelle sucht. Die Funktion `fun` wird dabei als `functionhandle` mit allen anderen wichtigen Daten an den Löser `newton_solver_...` übergeben.

Diese Funktion ruft `f_nonlin_...` auf. Diese Funktion hat alle nichtlinearen Teile der Gleichung gesammelt und führt dabei eine Approximation durch.

Nach dem Lösen werden die Lösungen in den zuvor definierten Lösungsmatrizen abgespeichert.

3 Löser

Die Löser verwenden das Newtonverfahren und nutzen dabei die Struktur von `fun` aus um die Jacobimatrizen zu berechnen. Sie wenden das Verfahren iterativ an, bis die Lösung die gewünschte Toleranz unterschreitet.

4 Potentiale

Die Ableitungen der zerlegten Potentiale sind als eigene Funktionen gegeben. Auch die zweite Ableitung der Potentiale ist wegen des Newtonverfahrens gegeben. Die Potentiale selbst sind unzerlegt gegeben.

`Final` plottet das System noch Massen und Energieeigenschaften. Dabei werden Quadraturfunktionen `integral_approx1d` und `integral_approx2d` verwendet, welche auf einfache Quadratur zurückgreift.

5 Feinere Diskretisierungen

Die feineren Löser haben eigene Skripte, denen ein `F` (feinere Zeitdiskretisierung) bzw. zwei `F` (feinere Zeit und Raumdiskretisierung) im Namen angehängt wurde (z.B. `time_stepping_AC!F`). Sie funktionieren analog, bilden jedoch größere Matrizen. Die feineren Schritte lassen sich mit den Parametern `k` und `l` steuern. Auch hier besitzt das System eigene Löser, die das Newton Verfahren speziell auf dieses Problem anwenden. Abgespeichert werden dabei neben den Lösungen zu den Zeitpunkten t^n auch alle Zwischenschritte.

Die unterschiedlichen Gitter erhält man in den jeweiligen Abschnitten vom Code. Bei dem `FF`-Codes werden weiterhin zwei neue Matrizen gebildet. Die Funktion `assembly_interpolmatrix` bildet eine Matrix, womit aus den Anfangswerten in Ω passende Anfangswerte für das unterschiedlich diskretisierte Γ gewonnen werden kann.

Die Funktion `assembly_tracematrix` bildet dabei die Matrix welche aus dem Spuoperator folgt. Diese Funktion ersetzt das zuvor verwendete `assembly_B` und übersetzt von der Diskretisierung im Inneren zur Diskretisierung am Rand.